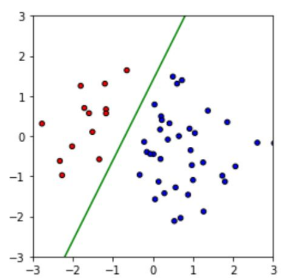
Perceptrons

Basic Linear Classifiers

- Assumes 2 classes of labels (**binary classification**)
  - Will work to recognize if diabetes or not
  - Will *not* work to recognize 10 handwritten digits
  - Looking ahead: will see how to "spoof" multi-class classifiers from binary classifiers
- Assumes a **linear decision boundary**
  - Looking ahead: will see how to manipulate linear classifiers to get arbitrary decision boundaries

Linear Classifiers

- Training: find a dividing "hyperplane" between two classes
- Testing: check which side of hyperplane the new point falls

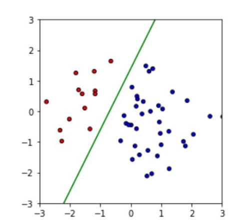


There are several algorithms to learn linear classifiers

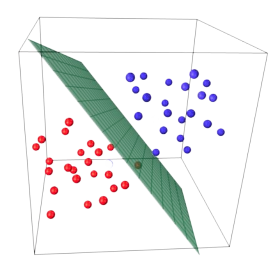
Hyperplane

A hyperplane in $\mathbb{R}^n$ is an n-1 dimensional subspace

A hyperplane in $\mathbb{R}^1$ is a point



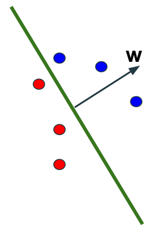
A hyperplane in $\mathbb{R}^2$ is a line



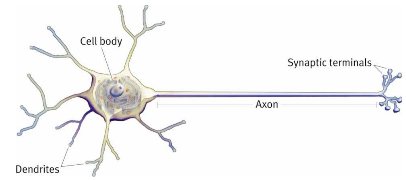
A hyperplane in $\mathbb{R}^3$ is a 2D plane

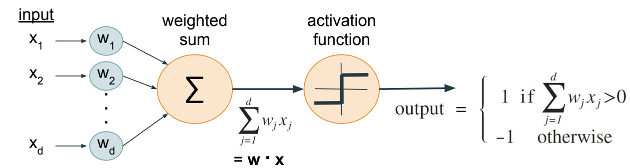
What is a hyperplane?

- Parameterized by a “weight” vector **w** *orthogonal* to the hyperplane, centered at origin
- What is the dimensionality of **w** in an *n*-dimensional space?
- What range is
  - The dot product of **w** with any of the blue points?
  - The dot product of **w** with any of the red points?



Perceptron Motivation





Perceptron Learning Algorithm

Two classes: one is +1 and the other is -1
Training data comes as vectors ***x*** and labels *y*

Start with vector **w** = all zeros

1. For each training datapoint ***x*** with label *y*:
   - If **w**·***x*** > 0 and *y* = +1, do nothing
   - If **w**·***x*** < 0 and *y* = -1, do nothing
   - If **w**·***x*** ≤ 0 and *y* = +1, **w** = **w** + ***x***
   - If **w**·***x*** ≥ 0 and *y* = -1, **w** = **w** - ***x***

Each **w** update *rotates* the hyperplane

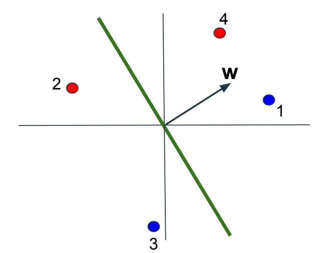
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

Perceptron Algorithm In Action

Two classes: one is +1 and the other is -1
Training data comes as vectors ***x*** and labels *y*

Start with vector **w** = all zeros

1. For each training datapoint ***x*** with label *y*:
   - If **w**·***x*** > 0 and *y* = +1, do nothing
   - If **w**·***x*** < 0 and *y* = -1, do nothing
   - If **w**·***x*** ≤ 0 and *y* = +1, **w** = **w** + ***x***
   - If **w**·***x*** ≥ 0 and *y* = -1, **w** = **w** - ***x***



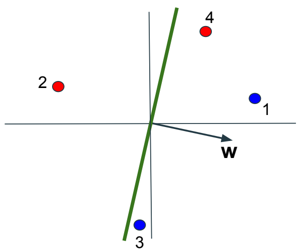
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

Perceptron Algorithm In Action

Two classes: one is +1 and the other is -1
Training data comes as vectors ***x*** and labels *y*

Start with vector **w** = all zeros

1. For each training datapoint ***x*** with label *y*:
   - If **w**·***x*** > 0 and *y* = +1, do nothing
   - If **w**·***x*** < 0 and *y* = -1, do nothing
   - If **w**·***x*** ≤ 0 and *y* = +1, **w** = **w** + ***x***
   - If **w**·***x*** ≥ 0 and *y* = -1, **w** = **w** - ***x***
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

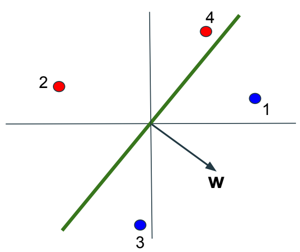


Perceptron Algorithm In Action

Two classes: one is +1 and the other is -1
Training data comes as vectors ***x*** and labels *y*

Start with vector **w** = all zeros

1. For each training datapoint ***x*** with label *y*:
   - If **w**·***x*** > 0 and *y* = +1, do nothing
   - If **w**·***x*** < 0 and *y* = -1, do nothing
   - If **w**·***x*** ≤ 0 and *y* = +1, **w** = **w** + ***x***
   - If **w**·***x*** ≥ 0 and *y* = -1, **w** = **w** - ***x***
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

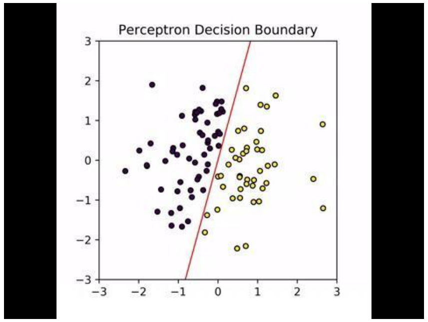


Perceptron Algorithm - Condensed Pseudocode

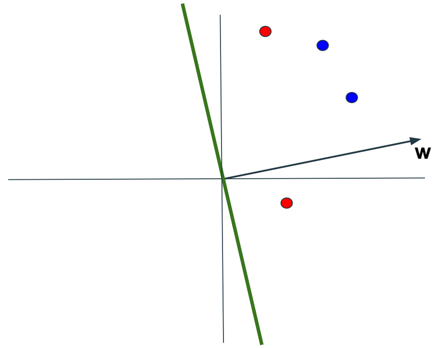
Start with vector **w** = all zeros

1. For each training datapoint ***x*** with label *y*:
   - If **w**·***x*** > 0 and y_i = +1, do nothing
   - If **w**·***x*** < 0 and y_i = -1, do nothing
   - If *y* * (**w**·***x***) > 0, do nothing
   - If **w**·***x*** ≤ 0 and *y* = +1, **w** = **w** + ***x***
   - If **w**·***x*** ≥ 0 and *y* = -1, **w** = **w** - ***x***
   - If *y* * (**w**·***x***) ≤ 0, **w** = **w** + *y* ***x***
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

Perceptron Algorithm In Action

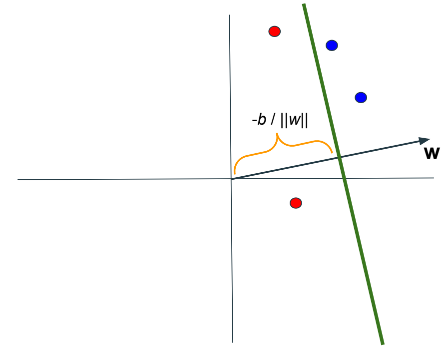


What if the hyperplane is not centered at the origin?

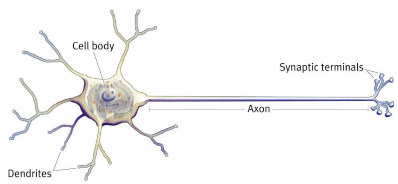


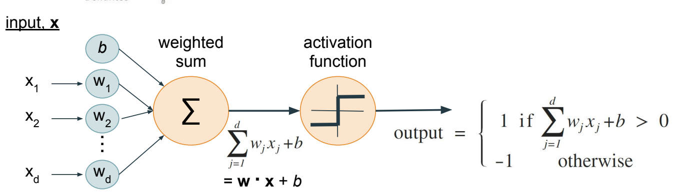
What if the hyperplane is not centered at the origin?

w·***x*** + b = 0 represents a hyperplane orthogonal to **w**, translated by $-b$ / $||w||$ in the direction of **w**



Perceptron Motivation





Perceptron Learning Algorithm

Two classes: one is +1 and the other is -1
Training data comes as vectors ***x*** and labels y

Each **w** update *rotates* the hyperplane

Start with vector **w** = all zeros

1. For each training datapoint ***x*** with label y:
   - If **w**·***x*** > 0 and y = +1, do nothing
   - If **w**·***x*** < 0 and y = -1, do nothing
   - If **w**·***x*** ≤ 0 and y = +1, **w** = **w** + ***x***
   - If **w**·***x*** ≥ 0 and y = -1, **w** = **w** - ***x***
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

Perceptron Learning Algorithm with bias term

Two classes: one is +1 and the other is -1
Training data comes as vectors $\boldsymbol{x}$ and labels y

Start with vector $\mathbf{w}$ = all zeros, and a bias term $b = 0$

1. For each training datapoint $\boldsymbol{x}$ with label y:
   - If $\mathbf{w}\cdot\boldsymbol{x} + b > 0$ and $y = +1$, do nothing
   - If $\mathbf{w}\cdot\boldsymbol{x} + b < 0$ and $y = -1$, do nothing
   - If $\mathbf{w}\cdot\boldsymbol{x} + b \leq 0$ and $y = +1$, $\mathbf{w} = \mathbf{w} + \boldsymbol{x}$ and $b = b+1$
   - If $\mathbf{w}\cdot\boldsymbol{x} + b \geq 0$ and $y = -1$, $\mathbf{w} = \mathbf{w} - \boldsymbol{x}$ and $b = b-1$
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

Each $\mathbf{w}$ update *rotates* the hyperplane

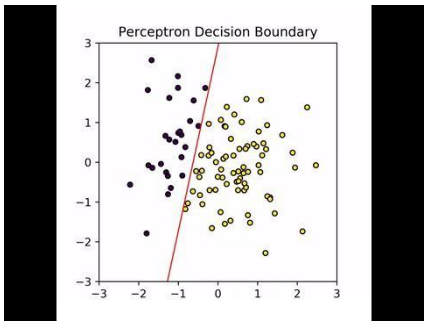
Each b update *translates* the hyperplane

Perceptron Algorithm with bias term Condensed

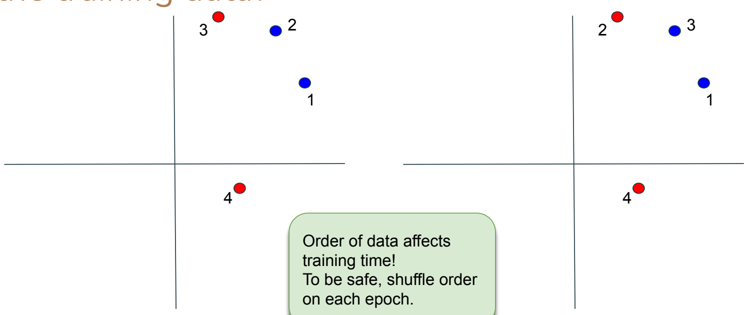
Start with vector $\mathbf{w}$ = all zeros, and a bias term $b = 0$

1. For each training datapoint $\boldsymbol{x}$ with label y:
   - If $\mathbf{w}\cdot\boldsymbol{x} + b > 0$ and $y = +1$, do nothing
   - If $\mathbf{w}\cdot\boldsymbol{x} + b < 0$ and $y = -1$, do nothing
   - If $y * (\mathbf{w}\cdot\boldsymbol{x} + b) > 0$, do nothing
   - If $\mathbf{w}\cdot\boldsymbol{x} + b \leq 0$ and $y = +1$, $\mathbf{w} = \mathbf{w} + \boldsymbol{x}$ and $b = b+1$
   - If $\mathbf{w}\cdot\boldsymbol{x} + b \geq 0$ and $y = -1$, $\mathbf{w} = \mathbf{w} - \boldsymbol{x}$ and $b = b-1$
   - If $y * (\mathbf{w}\cdot\boldsymbol{x} + b) \leq 0$, $\mathbf{w} = \mathbf{w} + y\,\boldsymbol{x}$ and $b = b+y$
2. Repeat step 1 until no more misclassified datapoints, or until *num_epochs* (where the number of epochs is a hyperparameter)

Perceptron Algorithm with bias term in Action

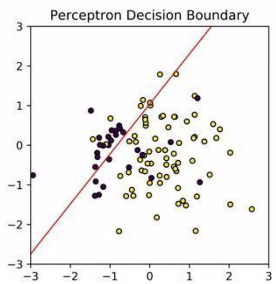


What happens if we change the order of the training data?



Order of data affects training time!
To be safe, shuffle order on each epoch.

Perceptron Algorithm - no Linear Boundary



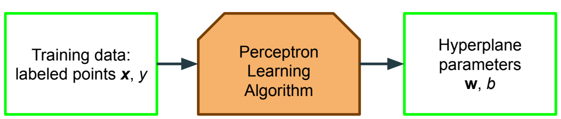
Testing

- Once the perceptron has been trained and the parameters **w** and *b* (i.e., the hyperplane) have been learned, we predict the class of a new datapoint **x** by determining which side of the hyperplane it falls on, i.e., by computing the weighted sum (i.e., dot product) followed by the activation function:

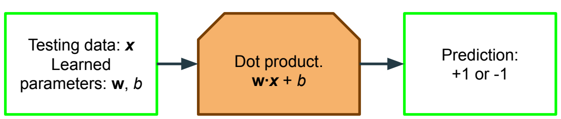
$$\text{predicted class} = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

Recap

- Training



- Testing



Complexity of Perceptron

- Training (as a function of *n* datapoints, *d* dimensions, and number of epochs)

- Testing

What does the trained hyperplane give us?

- Most importantly: a classifier to predict labels for new datapoints
- Also indicates which features are most important for each label

SPAM Email

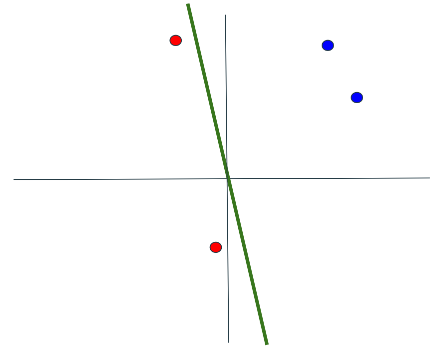
- Given dataset of email messages, where each feature is a word and the value is the number of times a message contains that word
- Train perceptron to classify SPAM messages vs non-SPAM (HAM) messages
- Resulting **w** shows which dimensions (aka features aka words) are most indicative of SPAM and HAM

Sentiment analysis

- Given dataset of movie/product/restaurant reviews, where each feature is a word and the value is the number of times a review uses that word
- Train perceptron to classify sentiment (positive or negative)
- Resulting **w** shows which dimensions (aka features aka words) are most indicative of positive or negative sentiment

Danger of Simple Perceptron

- Last few points have **too much** influence
- May result in a hyperplane that's "bad" even if it separates the training data



Solution 1: Voted Perceptron

- Training: Cache every hyperplane seen during training history, i.e., store every **w** and *b* and the number of times it occurs
- Testing: Given a new point ***x***, have every one of these cached hyperplanes vote with the number of times it occurs

Problem:
(1) Need to store 1000s of hyperplanes after training
(2) Testing time goes up drastically

Solution 2: Averaged Perceptron

Idea:
During training, compute the *average* hyperplane.
During testing, use this *average* hyperplane to classify a new point.

- Training: Rather than store every intermediate hyperplane seen during training (too expensive), instead keep track of a running sum of each hyperplane, i.e., a running sum of each **w** and *b*

$$\mathbf{u} = \mathbf{u} + \mathbf{w}$$
$$\beta = \beta + b$$

- At the end of training, compute the parameters of the average hyperplane:

$$\mathbf{u} = \mathbf{u} \,/\, (n^*epochs)$$
$$\beta = \beta \,/\, (n^*epochs)$$

- Testing: Given a new point ***x***, use the *average* hyperplane (based on **u** and **β**) to classify the point